WINDOWS 汇编语言及系统程序设计

[澳] Barry Kauler 著

张豫夫　曹建文　译　裘宗燕　审校

北京大学出版社

[澳] Barry Kauler 著

WINDOWS 汇编语言及系统程序

WINDOWS 汇编语言及系统程序设计

〔澳〕Barry Kauler 著

张豫夫　曹建文　译
裘宗燕　　审校

北京大学出版社
北　京

著作权合同登记　图字:01-95-511 号

内 容 简 介

本书是首册专门讲述 Windows 汇编语言程序设计的书籍。全书以许多简单的示例，说明进行 Windows 汇编语言程序设计是如何容易。书中介绍了面向对象设计方法的概念和实质，讲述了如何用汇编语言编写 Windows 下的面向对象程序。书中还介绍了 286/386/486 CPU 体系结构的高级特性，如保护模式，以及这些特性对程序设计的影响。同时也介绍了 Windows 的内部体系结构，并给出大量未公开的技术资料，对一些可能出现技术错误的细节也做了深入的探讨。书中还列出了一些难以找到的系统程序设计所需的信息，如 DPMI。

本书可帮助只有 DOS 汇编语言经验的读者迅速掌握 Windows 汇编语言程序设计技巧。

与本书配套的磁盘中有书中全部示例程序代码、一些难以找到的信息及一些实用工具。读者可与出版社联系购买。

图书在版编目（CIP）数据

Windows 汇编语言及系统程序设计＝WINDOWS ASSEMBLY LANGUAGE AND SYSTEM PROGRAMMING/（美）Barry Kauler 著；张豫夫，曹建文译．—北京：北京大学出版社，1995.9

ISBN 7-301-02915-2

Ⅰ.W… Ⅰ.①B… ②张… ③曹… Ⅱ.汇编语言-程序设计 N.TP312

本书原版英文版由 Prentice Hall of Australia Pty. Limited 出版，版权归 Barry Kauler 所有（Copyright ©1993 by Barry Kauler）。

本书中文版版权由 Prentice Hall of Australia Pty. Limited 授予北京大学出版社独家出版。未经出版者书面允许，不得以任何形式复制或抄袭本书内容。

版权所有，侵权必究。

书　　名：WINDOWS 汇编语言及系统程序设计
著作责任者：张豫夫　曹建文 译
责 任 编 辑：胡　燕
标 准 书 号：ISBN 7-301-02915-2/TP・270
出　版　者：北京大学出版社
地　　址：北京市海淀区中关村北京大学校内　100871
电　　话：出版部 2502015　发行部 2559712　编辑部 2502032
排　印　者：北京蓝地公司激光照排　北京飞达印刷厂印刷
发　行　者：北京大学出版社
经　销　者：新华书店
787×1092 毫米　16 开本　16.5 印张　420 千字
1995 年 11 月第一版　1995 年 11 月第一次印刷
定　　价：34 元

前　言

一本书的序，或者前言，通常都是在全书完成后才写，作为一本书的结束。在作者头脑中，它的作用与索引很类似。

我也不例外，也是这样做的，虽然我已经把这本书的索引尽量做得完善些，也希望读者能够这样认为。

但是这个前言却绝不是按照做索引的方式，简单而机械地生成出来的。相反地，我试图对全书做一个完美的总结，并分门别类讨论其中的问题，以期使那些随便翻翻这本书的人立刻想要买它，而一旦买了它就急于想知道书中到底讲了些什么。

我面临的一个问题是难以将本书划入某一类，或许更正确的是应当将它分割开，划入若干个不同的类中。问题的原因是本书的独特性，至少在本书编写的时候，还不存在任何其他的关于 Windows 汇编程序语言的书籍，仅仅在 Microsoft 的 Windows 软件开发工具包（Software Development Kit，SDK）手册《程序员参考手册》第四卷中有非常简短的一章与这个题目有关。

进一步的，也没有任何一本书讲述如此复杂的系统程序设计，讲述这些读者需要知道的，躲在 Windows 表面现象背后的实时、实模式、以及直接硬件操作等问题。

现在让我来分别阐述各个有关方面，以便读者能够确定哪些是更令自己感兴趣的东西。

1. Windows 汇编语言编码

首先是关于学习使用汇编语言为 Windows 编写程序的问题。由于几乎找不到什么已经发表的东西，大多数程序员完全不知道从何处开始下手。许多人相信 Windows 汇编语言程序设计是非常困难的。这或许是受了报界人士评论的影响，或者是由于查阅了 SDK 所附文档而产生的认识。

本书将要教授 Windows 汇编语言。假定读者只是熟悉 DOS 的汇编。这里要说明 Windows 汇编程序设计并不难，事实上这样做程序设计与用 C 语言或者其他高级语言编程一样容易。

因此，本书首先是一本关于 Windows 汇编语言编程的入门教科书，我们要清除其中的神秘性。

2. Windows 入门

要做 Windows 汇编语言编程，首先需要了解什么是 Windows 程序设计。除了书中关于汇编语言的内容外，对于那些只了解 DOS 的读者，本书也是学习 Windows 体系结构和程序设计的一本材料。因此从第二个方面说，本书不仅是关于 Windows 汇编语言程序设计，而且是关于一般的 Windows 体系结构和程序设计的入门书。

3. 高级体系结构

本书中所讲述的基本知识可以给读者日后的发展提供一个坚实的基础。对于 PC 和 CPU 体系结构特征的详细了解可以使读者更深刻地理解硬件是如何与操作系统、应用软件相互作用的。关于 286/386/486 的高级特征，特别是保护模式，向程序员和工程人员提出了全新的挑战。本书的另一个特点在于它是一本有关新型 CPU 和 PC 硬件体系结构的教程。

4. 去除 OOP 的神秘性

虽然我们的杂志报纸中大量地登载着“面向对象的程序设计”的字样，充斥着有关面向对象的产品与工具的无穷无尽的市场宣传，但是对于大多数程序员、工程人员和业余爱好者而言，OOP 仍然是一个很神秘的论题。即使是那些由于新近获得了 Boland 或 Microsoft 的 C++，或者其他类似产品而感到自豪的人们，也并不完全清楚他们得到了什么，以及如何最大限度地利用这些东西。

OOP 的教材常常使人感到困惑，而我在这里用非常简单的方法完全去除了 OOP 的神秘性。因此本书的另一个特点是介绍了高级的软件体系结构，特别是 OOP。

5. 汇编语言 OOP

本书中不仅要介绍 OOP，而且也要介绍汇编语言 OOP 的问题，并且要展示如何能编出一些十来行的完整的 Windows 程序。没有任何人在其他地方做过这样激动人心的事情。我不仅要说明如何编写独立的汇编语言 OO 程序，还要说明如何将它们与面向对象的高级语言（特别是 C++）接口。由此可见，本书的又一个特点是：它深入地介绍了 Windows 的汇编语言面向对象的程序设计。

6. 系统程序设计

Windows 程序员从 Microsoft 的 SDK、各种语言及其支持文档中获得的信息受到严格的控制。那些希望更详细地了解操作系统及其与体系结构的关系的人们不得不到处寻找。有关的信息可能在某些隐匿处找到，也可能需要花费大量的金钱，或者通过与 Microsoft 的特殊关系，（例如被批准为独立软件供应商，ISV）。在本书中，我列出了那些最有用的系统级服务的有关信息。例如 DPMI 服务（DOS 保护模式中断服务），以及其他很少发表的 Windows 函数和 DOS 服务。本书的另一个特点是介绍了 Windows 系统程序设计，是一本有关未公布的底层服务的参考手册。

7. MSAM 和 TASM

在开发本书中的代码时我使用了 Microsoft 的 MASM，也使用了 Borland 公司的 TASM。书中包含了有关 MASM 第 6.0+ 版和 TASM 第 3.0+ 版那些最新特征的讨论和一些实用性代码。这些新特征仅仅通过手册是很难掌握的。本书对于这些官方的手册是一个补充，对手册中提供的材料做了清晰的解释。特别的，由于 TASM 的 OO 倾向，它的手册中相当一部分内容很难理解，本书在这些方面将对读者有所帮助。

此为试读，需要完整PDF请访问：www.ertongbook.com

8. 配套磁盘

与本书配套的磁盘也非常值得收藏。在这个盘里读者不仅能找到书中讨论的所有例子的源代码，还能找到一些额外的文档和实用工具。盘中还存放了若干绝妙的商业演示软件，例如 Eclectic Software 公司的 WINTOASM 反汇编程序。

这个磁盘中包含了大量读者需要花费许多时间去寻找的，或者根本就不知道的信息。因此它本身也是极其宝贵的。

因此，你有八个充分的理由来购买这本书。读了这本书后，你可以去做那些其他 Windows 程序员甚至都不知道从何处下手的事情，比如在第九章中讲述的直接访问存储器和 I/O，第十章讲的与实时事件同步的问题，或者第十一章讲解的关于在保护模式与实模式之间的交互等。这些信息都是 Windows 程序员急切地想弄清楚的。

但是另一方面，本书也不只是为那些专业程序员服务的。教师们应当严肃地问一问自己，在他们当前的课程中包含了多少本书中讨论的概念。如果你的学校里有汇编语言课程，或者系统程序设计课程，或者微型计算机体系结构课程，那么这些课程的内容是否真的讲到了点上，可能不少学校还在教授昨天的技术，使用过时的教科书。

我也同样重视来自那些业余爱好者和计算机迷的极大热情，特别是那些“bit 狂”，以及那些想用 Windows 去做那种从来也没有人设想它能够做的事情的人们。

你实际上使用哪一种语言这一点并不重要，如果你的 Windows 编译器支持直接插入汇编，你可以用它。甚至也可以使用那些具有自己的汇编指令模拟机制的编译器。这方面的另一个极端是使用独立的汇编程序。

这里我想给出一个最后的，但绝不是最不重要的声明，在汇编语言的层次上为 Windows 编程不但是实用的，而且是极有趣的。请允许我再加上一句，我期待着 Windows 迷们的一个全新时代的到来。

鸣谢

感谢我的家庭，感谢你们的鼓励。

感谢我的 Edith Cowan 大学的同事们，感谢你们的建议和支持。

感谢我的计算机，感谢你承受了我对你的无穷无尽的折磨。

Barry Kauler

目　录

第一章　CPU 体系结构

1.1　引　　言

本章包含一些具有相当难度的理论，因此读者可不以此章作为学习的开端，以免半途而废。

阅读此章时不要考虑过多的细节，而要注重其中主要的思想。一种有益的办法是做快速浏览，直到需要使用本章理论的章节时，再返回来重新阅读其中的有关细节。书后的索引是相当全面的，可以帮助读者查找所需要的细节。

·预备知识

如果读者发现要消化的概念仍然太多，我建议你去读一些更基础的东西。我认为最合适的是我的书：《PC 体系结构与汇编语言》，这是一本相当厚的从头开始介绍 PC 机体系结构和汇编语言程序设计的书，并附有大量练习。它是很受欢迎的，并被一些大学做为教科书使用。

上面提到的那本书主要涉及 Intel 的 8088 和 8086 程序设计，本书讲述 80286，80386，80486 及以上 CPU，应该注意到，后来的 CPU 是向下兼容的，也就是说在它们上而可运行早期 CPU 上的软件；而反过来在早期的 CPU 上运行为 80286 以上 CPU 设计的软件则不一定成功。

本章要指出 Intel 家族中各类 CPU 之间的主要差别。

随着本书的展开，我们将进入程序设计领域，那里需要用到本章所讲的有关体系结构的概念。由于本书讲述的是 Windows 程序设计，因而也可以作为 Windows 的入门教材。但同其他 Windows 程序设计的书相比，本书的重点放在更基础的层面上。

有了本章中讲述的基础知识以后，读者可容易地在 Windows 下实现各种技巧，例如直接键盘输入、直接视频输出和通过中断进行信号处理等。

1.2　常规内存和扩充内存

扩充内存（extended memory）是内存 1 兆（M）地址界限以上的部分，常规内存则低于 1M。扩展内存（expanded memory）是存储模块切换内存，它可以被映象到常规内存区。扩充内存的首 64K 字节有时被称为高端内存（high memory）。

·PC 内存映象

图 1.1 中的映象所示的是装入 DOS 后的内存映象，如果正在运行 Windows，情况会有所不同，并且该映象也随 Windows 是运行在标准模式还是增强模式下而不同。Windows NT（NT 代表 New Technology）根本不需要 DOS，因此它的映象也更不同，但是上述各种情况都以这里的映象作为出发点。

此为试读，需要完整PDF请访问：www.ertongbook.com

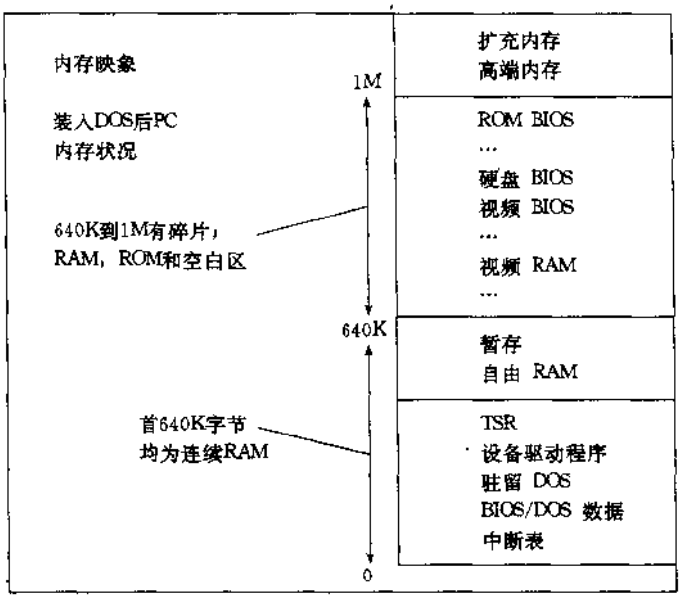

图 1.1

第十二章将深入到相当复杂的增强模式中去，在该模式下，图 1.1 中的基本内存映象可不再限于 0—1M 的实物理地址范围内。386 可以产生多个虚拟机（virtual machine），从表面上看，每个虚拟机都有 1M 地址空间，注意，虚拟机的地址称为虚拟地址（virtual address）。

不幸的是，PC 机的情况相当混乱。它诞生于 1980 年，那时它用的是只能显示字符文本的屏幕，卡式大容量磁带机（没有硬盘），没有实时时钟，只有 64K RAM 和 8088 CPU。多年来，各种特性不断地添加上去，操作系统和硬件也不断发展。

从 8080 继承来的一个令程序员头痛的问题是段。

1.2.1 段

早于 8088 的 CPU 片子（如 Intel 8008 和 Zilog Z80）都是 8 位的，即它们有 8 位宽的数据总线，而 8088 引进了 16 位体系结构（但是，8088 只有 16 位的内部数据通道，而外部数据总线只有 8 位。8086 与 8088 基本相同，但它有真正的 16 位外部数据总线）。

• Intel CPU 的历史

早期的 8 位CPU 有 16 位地址总线，用二进制来计算，其地址范围从 0 到 $2^{16}-1$，地址上限用二进制表示是 1111111111111111，用十六进制表示是 FFFF，用十进制表示是 64×1024−1=65535。我们通常称这一内存容量为 64K，其中 K 代表×1024（注意，1 兆是 1024×1024，所以 1M=1×1024×1024=1048576 字节）。

Intel 设计人员希望在 8086 家族的基础上增加内存容量以使其足够使用，因而为 8088 和 8086 提供了 20 位地址总线，2^{20}=1M。约 1 百万个字节的内存在当时是一个极大的数量。但到后来却变成了一个严重的局限，成了一个令人头痛的问题。另一个难题是如何设计寻址 1M 的芯片。

当时的设计人员希望能较容易地从 8 位 CPU 上移植软件。在早期 CPU 的内部有一个指令指针 IP，它是一个寄存器，指明下一条将要执行的指令的地址。此指针是 16 位的，与外部地址总线相配。

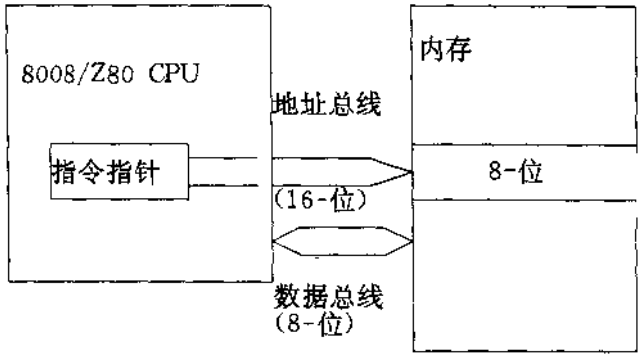

图 1.2

注意，每个内存单元都是 8 位（1 字节），每个单元都由一个唯一（不重复）的地址值标识。并且这一情况一直延续到最新具有 32 位或更宽数据总线的 Intel x86CPU。

·段的应用

关于段的问题。设计者引入了一个寄存器，称为段寄存器，利用它把由 IP 寻址的 64K 区域映象到 1M 范围的任何地方。代码仍然可以认为是在 64K 空间中，但实际上它已被透明地映象到段寄存器所指定的位置。这种策略的目的是兼容性，但是它也增加了复杂性，图 1.3 展示了这一映象过程。

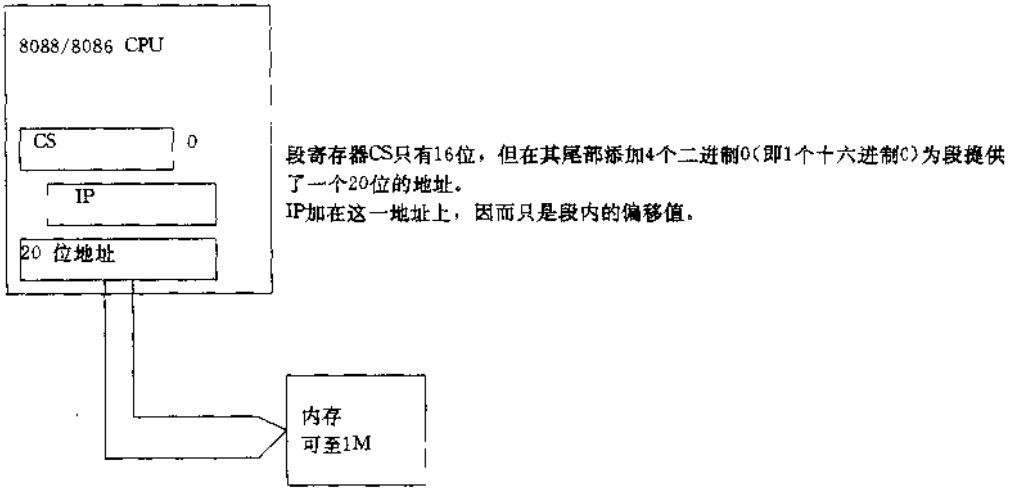

图 1.3

了解这个映象过程是绝对重要的。16 位的 IP 值加上 20 位的基地址值便给出了 CPU 取出下一条指令的 20 位地址值。因此，IP 值是段中的偏移值，正是这个原因使得一个段最大容量只能有 64K。

.COM 格式的可执行文件诞生于 8 位 CPU 时代，它被限制在 64K 的范围内。然而，后来设计者认识到、并通过再引入 3 个段寄存器来“解决”这一问题。这 3 个段寄存器是：DS（数据段）、SS（栈段）和 ES（附加段）。为了支持这些寄存器，设计者引入了 .EXE 可执行

文件结构，允许代码存储在用CS指明的段中，数据存储在由DS指明的一个段中，栈存储在由SS指明的一个段中，ES则指向一个可以由应用程序设计人员使用的段。尽管段的最大仍然只是64K，但大的程序可以有多个代码段和数据段。

与段有关的一个难题是段的64K限制。显然大的代码和数据可超出这一界限，问题也就产生了。另一个难题是为了兼容性。这个段的方案也传给了286/386等CPU。

1.2.2 实模式

8088/8086在我们目前称之为实模式（real mode）下操作，这意味着（同上文所解释的一样）段寄存器中存着实际地址值。286/386等芯片开机后也在实模式下运行，并且使用同样的20位段式寻址模式。为了兼容性，更先进的CPU也只能寻址到1M，额外的地址线未被激活。

1M限制是DOS和DOS应用程序的基本问题。但是也有例外，实模式寻址可以超出1M限制，有一个额外的64K字节，这64K被称为高端内存段（high memory segment）。看一下图1.4中寄存器的最大值便可以明白之所以有这个例外的原因：

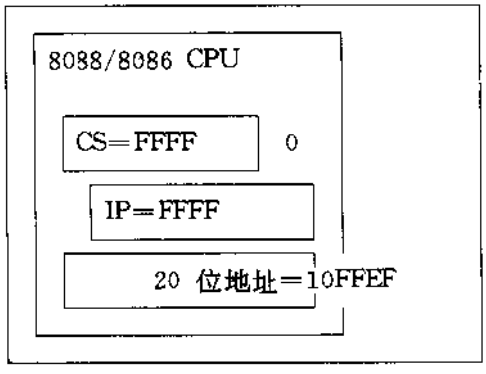

图 1.4

·高端内存

20位的上限值是十六进制的FFFFF（1M－1），但至少在理论上编移值IP使得寻址能够超出这一界限。要完成此功能，从物理上讲需要第21个地址位，8088/8086没有这一位，在286等芯片上该位也被禁止，但可以通过编写指令来激活286/386芯片上的第21位，从而访问超过1M的64K。

额外的64K是微乎其微的。286有24条地址线，理论上可有2^{24}字节，也就是16M字节的内存。386有32条地址线，可有4.3G内存空间。但是当CPU在实模式下运行时，这些内存都不能使用。

1.2.3 DOS实模式程序设计

DOS自身，DOS和BIOS I/O服务程序、DOS应用程序、设备驱动程序和常驻内存程序（TSR）都被设计成在实模式下工作的，这是因为它们都依赖段寄存器中的实地址值。

·在程序中使用段寄存器

现在考虑一个例子。一位程序员需要使用ES寄存器直接写视频RAM。视频RAM与其他RAM类似，只是无论把什么写入其中，它都将在屏幕上显示出来。程序员可将十六进制值

B800 装入 ES，将可寻址到 CGA 视频 RAM（即该段的 20 位始地址值为十六进制的 B8000）。在汇编语言中，程序员书写“MOV ES:[DI], AL”或“STOSB”指令将通用寄存器 AL 中的值存入地址 ES：DI 中。

注意，术语 ES：DI 指的是段：偏移值形式的地址。ES 是段、DI（或其他 16 位寄存器）中放着偏移值。当然，DI 和 AL 事先都应该装有值。

此处的要点是程序在 ES 中装入了一个实在的段地址值。对段寄存器中实值的依赖意味着 DOS 程序不能在 1M 以上工作。

·选择符

286/386 CPU 可以切换到保护模式（protected mode）的状态下，这一模式允许访问多至 16M 或更多的内存，但是实地址值需要从段寄存器中清除掉，代之以指到某些表格中的选择符（selector）或索引，表格中存着段的实地址值。

DOS 提供了一些切换进或切出保护模式的机制，这里介绍两种早期的 DOS 服务程序。

1.2.4 DOS 保护模式程序设计

这方面是一团乱麻，太多的问题，使人不知从何处开始。我曾经讨论过在保护模式下运行 DOS 应用程序方面的一些问题，以及 DOS 本身在这个方面的毛病。

·INT 15h

Microsoft 是逐渐扩展 DOS 的，首先增添的服务是通过 INT 15h 激活的一些功能。我们就从这里开始讨论。

这里的想法是提供从实模式到保护模式（或反过来）的切换方法，将代码在常规内存与扩充内存之间传送，并从将执行实模式程序切换到保护模式程序。

·INT 15h，AH＝88h，AH＝87h

INT 15h，AH＝88h 将告知有多少扩充内存，但不告知扩充内存是如何被使用的。这一功能简单地返回一个值到 AX 寄存器中。图 1.8 说明了 CPU 寄存器的情况。

1NT 15h，AH＝87h，在常规内存和扩充内存间移动一块数据。我们马上会想到这一功能可以应用到 TSR 管理程序中，用于将所有常驻内存程序搁置起来，并在需要时取回一个（确实有这样的管理程序可使用）。

·局部和全局描述符表

需要为寻址提供各种表格是保护模式，也是通常管理工作要处理的一个问题。段寄存器中不再含有实地址值，实地址值存放在局部描述符表（LDT）或全局描述符表（GDT）中。如果中断是由保护模式中的程序控制的，还需要一个中断描述符表（IDT）。再进一步，如果允许任务切换，还要为每个任务提供一个任务状态段（TSS）。

幸运的是，INT-15h，AH＝87h 使表的问题相当简单，所需要的只是一个 GDT，以使服务程序能够工作，并由应用程序来设置此 GDT：

CX：欲传送的字数

ES：ST：GDT 的物理地址（在常规内存中）

AH＝87h

·将 CPU 切换到保护模式下

DOS 服务程序在处理切换时对 286 和 386 有所不同。在 286 中需要将机器状态字寄存器

中的保护允许位置位，并设置描述符表，将GDT（全局描述符表）的地址装入GDT指针寄存器中。

在286和386中，进入保护模式时需要对机器状态字中相应的位置位。在386中还可以通过简单的相同置位返回实模式。而286没有返回实模式的机制，只能通过复位（reset）CPU这一非常慢的方法来完成，这个过程要几个毫秒。这也是286很快成为历史的一个原因。

·INT-15h，AH＝89h

INT-15h，AH＝89h真正将控制从实模式代码传送给保护模式代码。此时需要有一个IDT，这是因为如果PC继续操作可能产生硬件中断，而应用程序也可能需要产生软件中断。所以这一服务需要更多的管理工作。

我以前的书中列有关于从实模式传送代码到扩展模式，关于怎样设置GDT的练习题。为帮助读者了解GDT是什么样子，图1.5给出了一个INT-15h/87h所要求的基本的GDT。对于386系统，GDT的形式有些小的变动。

偏移值内容	
00-07h	保留(应为0)
08-0Fh	此GDT的描述符
10-17h	IDT的描述符
18-1Fh	DSR的描述符
20-27h	ES的描述符
28-2Fh	SS的描述符
30-37h	CS的描述符
38-3Fh	临时BIOS CS的描述符

图 1.5

·GDT的创建

从图1.5应当了解到，在调用INT-15h之前，应用程序需要先创建一个GDT，也许还有一个IDT，并将有关的地址值放入GDT中，而当服务实现向保护模式切换时，还要为DS，ES，SS和CS装入描述符到GDT中。

作为GDT索引的选择符（selector）将由DOS服务程序装入CPU内的DS、ES、SS和CS寄存器，GDT内保存真实地址。

正像我们所知道的那样，DOS在实模式下从磁盘上装入一个程序时，将这个程序置于自由内存中，并自动设置相应的DS，ES，SS和CS。但在保护模式下，段的实地址放在描述符表中，段寄存器中只有指向这个表的指针。一个很重要的问题是为什么要将实地址值放在表里，而不是CPU中。原因很简单，段寄存器是16位的，从而将地址限制在1M以内，而表中的段地址至少有24位，从而可提供至少16M的地址范围。

·编码的约束

从上文可知，能够编写出在保护模式下运行的DOS应用程序。DOS可将代码和数据装在1M以上空间，能切换到保护模式，以及在1M以上运行程序。

这里的基本要求是程序不应认为在段寄存器中存放的是实在的物理地址。同时也隐含着程序不能调用BIOS和DOS I/O服务程序（通常通过INT指令来调用），因为这些程序都是被设计成在实模式下运行的。设备驱动和常驻内存程序也存在同样的问题。一种可能的办法是切换回实模式去运行一个I/O服务程序或设备驱动程序，或者提供BIOS和DOS I/O服务程序的保护模式版本。这些在本书后面的章节中将进一步讨论。

1.3 286/386/486内部结构

此节将重点放在386上，因为286不久就将成为历史了。486可认为在功能上与386一样，只是更快一些。在写作本书时，586的信息较少，但同样可认为是一种快得多的向上兼容的386。

386，486和586之间在体系结构上面都有一些差别，这里将把重点放在基本的体系结构——386上。

·32位指令指针

几乎一切都变成了32位的。地址和数据总线是32位的。指令指针也发展成32位的（见图1.6），这意味着引入段寄存器的原始合理性已荡然无存。但是，段寄存器仍然在那里，仍然只有8位，为了兼容而留下的弊端依旧存在。

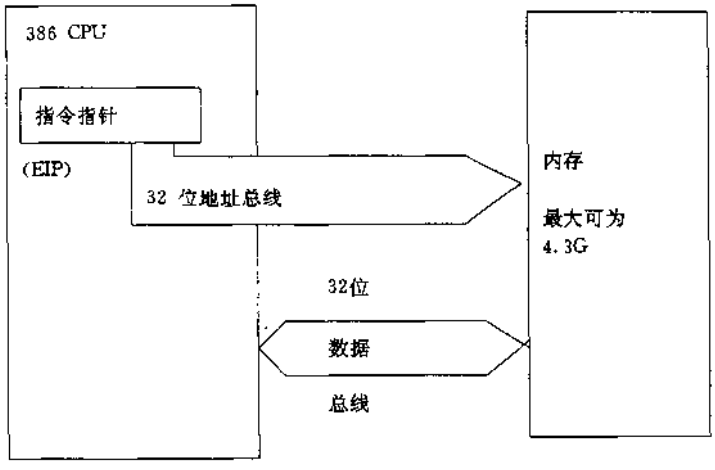

图 1.6

·内存和I/O地址空间

32位提供了极大的寻址能力：43亿字节（4.3G）。但必须注意，为了兼容，每个地址值只寻址内存中的8位[①]数据，尽管数据总线有32位。

I/O端口的寻址与88/86/286相同，使用物理地址总线的低16位，加上IOR和IOW控制线。16位允许有多至65536个I/O端口。应当注意到，I/O地址空间不包含在4.3G内存地

① 原文为8个字节，疑误——译者注。

此为试读，需要完整PDF请访问：www.ertongbook.com